data.table

New Developments

Tyson S. Barrett

rfinance(2024L)

Who Am I?

- Current "maintainer" of `data.table` (more on this at the end!)
- PhD in Quantitative Psychology (Bachelor's in Economics)
- `data.table` user since 2016, contributor since 2019
- Author/maintainer of 6 other R packages (3 on CRAN)
- Currently managing a team of researchers at Highmark Health (lots of big data wrangling and cleaning)
- CEO of Barrett Evaluation, LLC (big talk for I consult on projects with big data)

“dplyr will be the death of data.table”

An attendee said to Matt Dowle (creator of data.table) at an R Finance Conference a decade ago

```r
library(tidyverse)
df %>%
  filter(x == 1) %>%
  mutate(z = y * 2)
```

```r
library(data.table)
dt[x == 1][, z := y * 2]
```

Agenda

- Why use `data.table`?
- New developments!
  - New “management”
  - New features
  - New ways to engage

Why data.table ?

Concise syntax

Fast speed

Memory efficient

Careful API lifecycle management

Community

Feature rich

Matt's useR talk from 2014 https://youtu.be/qLrdYhizEMg?t=1m54s

Why data.table ?

- Concise syntax
- Fast speed
- Memory efficient
- Careful API lifecycle management
- Community
- Feature rich

Why data.table?

- Concise syntax
- Fast speed
- Memory efficient
- Careful API lifecycle management
- Open source
- Feature rich

```
dt[i, j, by]
```

- i: "WHERE" or "ORDER BY"
- j: "SELECT"
- by: "GROUP BY" or "PARTITION BY"

Why data.table?

Concise syntax

Fast speed

Memory efficient

```
dt[grp == “treatment”, new := mean(x), by = id]
dt[dt2, on = “id”]
dt[dt2, on = “id”, roll = TRUE] # rolling joins!
dt[, .N, by = id]
```

Why data.table ?

Concise syntax

Fast speed

Memory efficient

Careful API lifecycle m

Community

Feature rich

Responding to why `data.table` is so fast, Hadley Wickham:

"I think it's a relentless focus on performance across the entire package."

https://twitter.com/hadleywickham/status/1153850194892640256

Before the death of Twitter

Why data.table ?

Concise syntax

Fast speed

Memory efficient

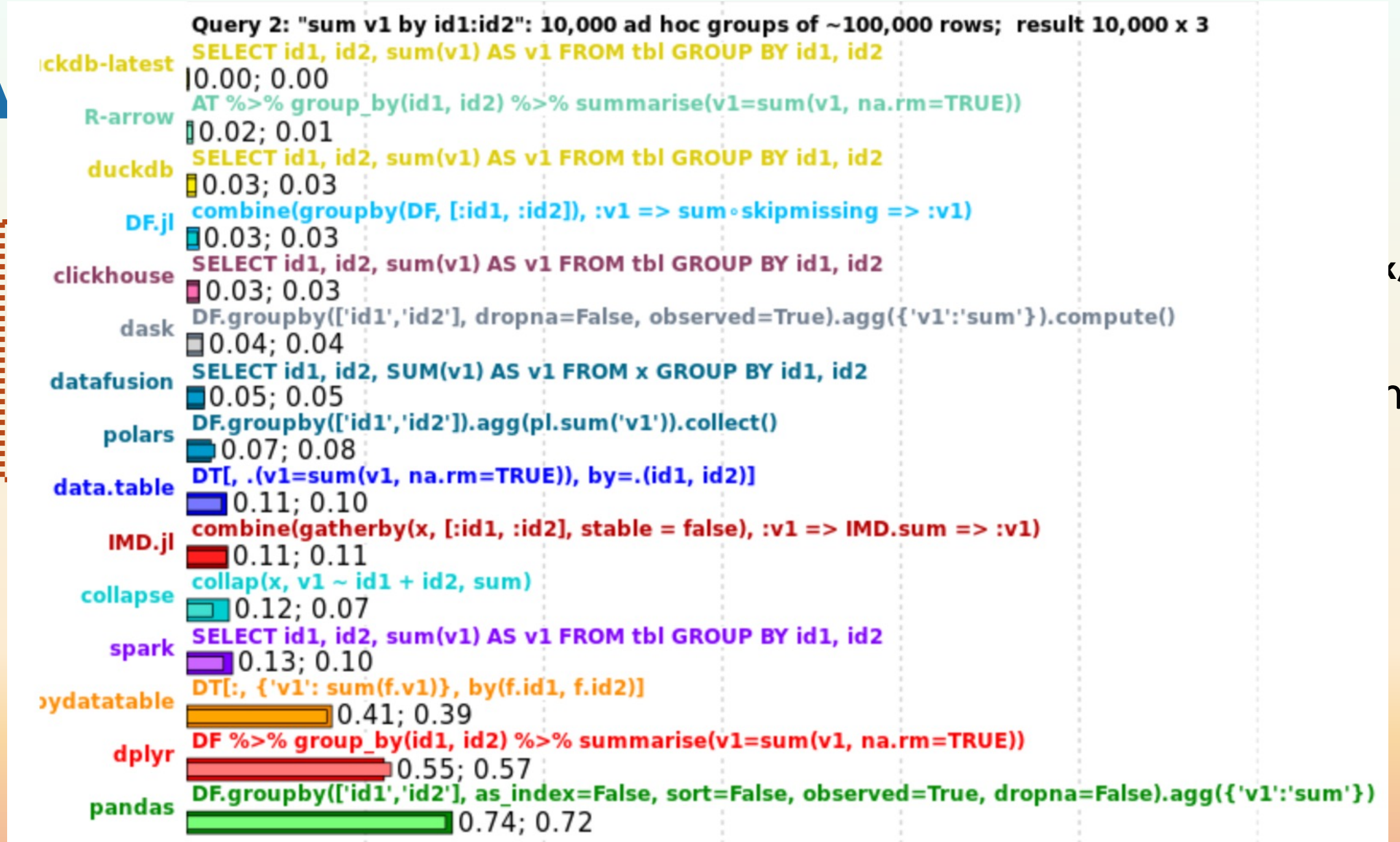
Careful API lifecycle management

Community

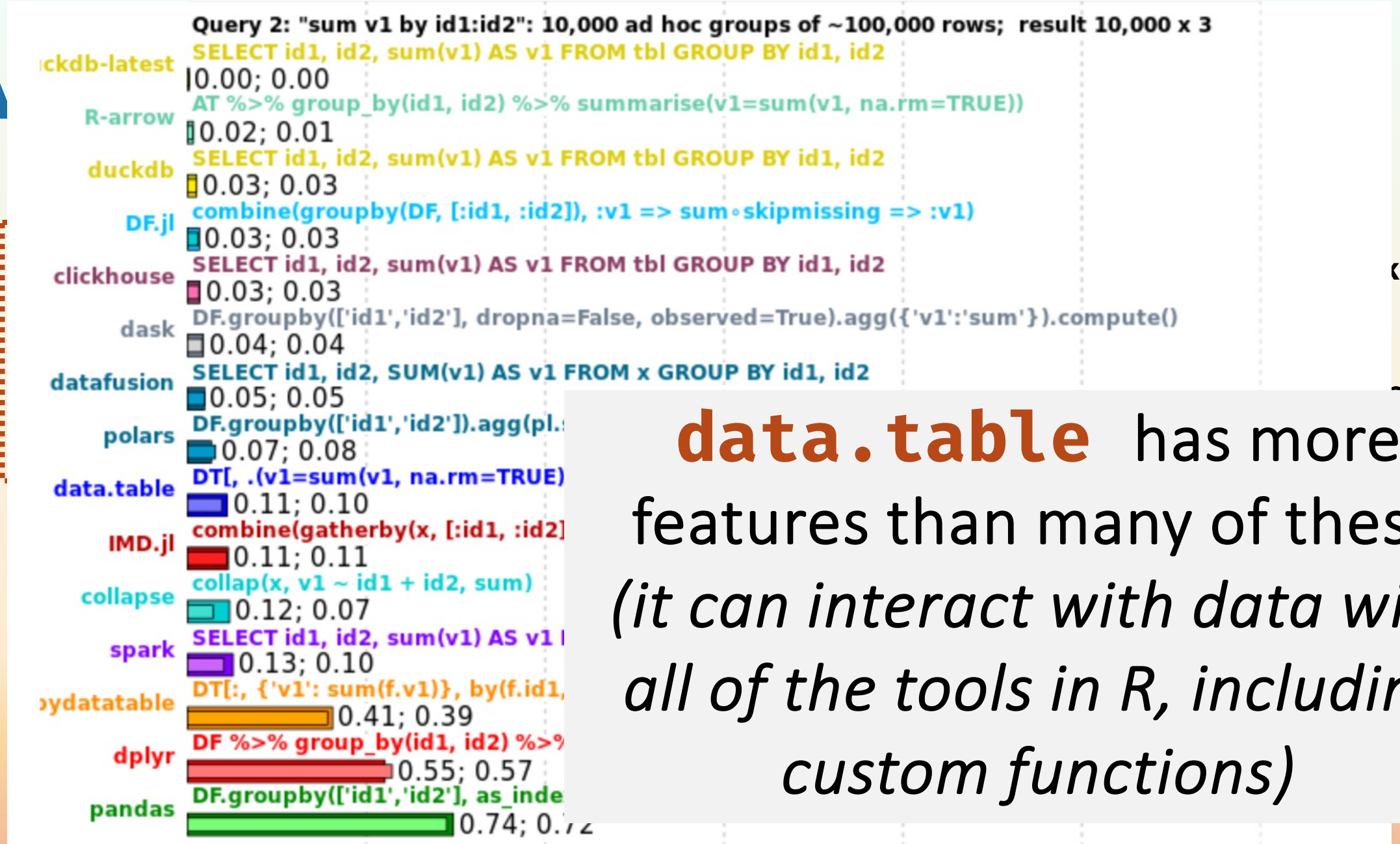
Feature rich

https://duckdblabs.github.io/db-benchmark/

It's consistently one of the top performing, even with major investments in tools like DuckDB and Polars


Query 2: "sum v1 by id1:id2": 10,000 ad hoc groups of ~100,000 rows; result 10,000 x 3
ckdb-latest
SELECT id1, id2, sum(v1) AS v1 FROM tbl GROUP BY id1, id2
0.00; 0.00
R-arrow
AT %>% group_by(id1, id2) %>% summarise(v1=sum(v1, na.rm=TRUE))
0.02; 0.01
duckdb
SELECT id1, id2, sum(v1) AS v1 FROM tbl GROUP BY id1, id2
0.03; 0.03
DF.jl
combine(groupby(DF, [:id1, :id2]), :v1 => sum∘skipmissing => :v1)
0.03; 0.03
clickhouse
SELECT id1, id2, sum(v1) AS v1 FROM tbl GROUP BY id1, id2
0.03; 0.03
dask
DF.groupby(['id1','id2'], dropna=False, observed=True).agg({'v1':'sum'}).compute()
0.04; 0.04
datafusion
SELECT id1, id2, SUM(v1) AS v1 FROM x GROUP BY id1, id2
0.05; 0.05
polars
DF.groupby(['id1','id2']).agg(pl.sum('v1')).collect()
0.07; 0.08
data.table
DT[, .(v1=sum(v1, na.rm=TRUE)), by=.(id1, id2)]
0.11; 0.10
IMD.jl
combine(gatherby(x, [:id1, :id2], stable = false), :v1 => IMD.sum => :v1)
0.11; 0.11
collapse
collap(x, v1 ~ id1 + id2, sum)
0.12; 0.07
spark
SELECT id1, id2, sum(v1) AS v1 FROM tbl GROUP BY id1, id2
0.13; 0.10
ydatatable
DT[:, {'v1': sum(f.v1)}, by(f.id1, f.id2)]
0.41; 0.39
dplyr
DF %>% group_by(id1, id2) %>% summarise(v1=sum(v1, na.rm=TRUE))
0.55; 0.57
pandas
DF.groupby(['id1','id2'], as_index=False, sort=False, observed=True, dropna=False).agg({'v1':'sum'})
0.74; 0.72

W



data.table has more features than many of these *(it can interact with data with all of the tools in R, including custom functions)*

Why data.table?

Concise syntax

Fast speed

Memory efficient

Careful API lifecycle management

Community

Feature rich

Thoughtful and careful so there are very few breaking changes

Can be used in production code safely

Matt's useR talk from 2014 https://youtu.be/qLrdYhizEMg?t=1m54s

New Developments!

- Grant from NSF (PI = Toby Hocking) to create new governance and support its development (NSF POSE program, project #2303612)
- Re-invigorated development and new features
- Ways to engage in development

New Developments!

- Grant from NSF (PI = Toby Hocking) to create new governance and support its development (NSF POSE program, project #2303612)

`https://github.com/Rdatatable/data.table/blob/master/GOVERNANCE.md`

Nev

- Gran ce and
  supp 612)

http E.md

Decision-making processes

Definition of Consensus

Most decisions in the project happen by Consensus, which means that no active people (typically Reviewers and/or Committers) have expressed major blocking concerns, in a public discussion (typically in a GitHub issue or pull request). In Consensus, non-response by inactive members indicates tacit agreement.

Pull Requests

A pull request can be merged by any committer, if there is one approving review, and Consensus from active Reviewers and Committers.

- approving review must come from someone other than the author of the PR.
- approving review ideally comes from a reviewer of the affected files.
- approving review can and often will be by the committer who merges the PR.

CRAN updates

- Regular CRAN releases should ideally occur twice per year, and can include new features.
- A hotfix/patch CRAN release should occur when CRAN asks for one, at which time the CRAN maintainer should post an issue on github, and ask others to help fix/prepare the release. It should not include new features.
- Both kinds of releases should be discussed in an issue, and the release should happen only if there is Consensus among active Reviewers and Committers.

A semi-democratic approach to dev

- Can become any role in data.table by submitting PR and enough votes from the community
- Can help shape the development of the package
- One aspect of the governance is the “what is possible for development” which can be updated

New Developments!

- Re-invigorated development and new features

data.table v1.15.0 (30 Jan 2024)

BREAKING CHANGE

1. `shift` and `nafill` will now raise error `input must not be matrix or array` when `matrix` or `array` is provided on input, rather than giving useless result, #5287. Thanks to @ethanbsmith for reporting.

NEW FEATURES

1. `nafill()` now applies `fill=` to the front/back of the vector when `type="locf|nocb"`, #3594. Thanks to @ben519 for the feature request. It also now returns a named object based on the input names. Note that if you are considering joining and then using `nafill(...,type='locf|nocb')` afterwards, please review `roll=` / `rollends=` which should achieve the same result in one step more efficiently. `nafill()` is for when filling-while-joining (i.e. `roll=` / `rollends=` / `nomatch=`) cannot be applied.

2. `mean(na.rm=TRUE)` by group is now GForce optimized, #4849. Thanks to the h2oai/db-benchmark project for spotting this issue. The 1 billion row example in the issue shows 48s reduced to 14s. The optimization also applies to type `integer64` resulting in a difference to the `bit64::mean.integer64` method: `data.table` returns a `double` result whereas `bit64` rounds the mean to the nearest integer.

3. `fwrite()` now writes UTF-8 or native csv files by specifying the `encoding=` argument, #1770. Thanks to @shrektan for the request and the PR.

4. `data.table()` no longer fills empty vectors with `NA` with warning. Instead a 0-row `data.table` is returned, #3727. Since `data.table()` is used internally by `.()`, this brings the following examples in line with expectations in most cases. Thanks to @shrektan for the suggestion and PR.

41 new features!

And several fixes and speed ups

New Developments!

- Re-invigorated development and new features

33. `DT[, let(...)]` is a new alias for the functional form of `:=`; i.e. `DT[, ':='(...)]`, #3795. Thanks to Elio Campitelli for requesting, and Benjamin Schwendinger for the PR.

```
DT = data.table(A=1:2)
DT[, let(B=3:4, C=letters[1:2])]
DT
#        A     B      C
#    <int> <int> <char>
# 1:     1     3      a
# 2:     2     4      b
```

10. A new interface for *programming on data.table* has been added, closing #2655 and many other linked issues. It is built using base R's `substitute`-like interface via a new `env` argument to `[.data.table`. For details see the new vignette *programming on data.table*, and the new `?substitute2` manual page. Thanks to numerous users for filing requests, and Jan Gorecki for implementing.

```
DT = data.table(x = 1:5, y = 5:1)

# parameters
in_col_name = "x"
fun = "sum"
fun_arg1 = "na.rm"
fun_arg1val = TRUE
out_col_name = "sum_x"

# parameterized query
#DT[, .(out_col_name = fun(in_col_name, fun_arg1=fun_arg1val))]

# desired query
DT[, .(sum_x = sum(x, na.rm=TRUE))]

# new interface
DT[, .(out_col_name = fun(in_col_name, fun_arg1=fun_arg1val)),
  env = list(
    in_col_name = "x",
    fun = "sum",
    fun_arg1 = "na.rm",
    fun_arg1val = TRUE,
    out_col_name = "sum_x"
  )]
```

eenshot

New Developments!

- Re-invigorated development and new features

17. `data.table` printing now supports customizable methods for both columns and list column row items, part of #1523. `format_col` is S3-generic for customizing how to print whole columns and by default defers to the S3 `format` method for the column's class if one exists; e.g. `format.sfc` for geometry columns from the `sf` package, #2273. Similarly, `format_list_item` is S3-generic for customizing how to print each row of list columns (which lack a format method at a column level) and also by default defers to the S3 `format` method for that item's class if one exists. Thanks to @mllg who initially filed #3338 with the seed of the idea, @franknarf1 who earlier suggested the idea of providing custom formatters, @fparages who submitted a patch to improve the printing of timezones for #2842, @RichardRedding for pointing out an error relating to printing wide `expression` columns in #3011, @JoshOBrien for improving the output for geometry columns, and @MichaelChirico for implementing. See `?print.data.table` for examples.

New Developments!

- Ways to engage in development

1 GitHub Issue Tracker

New Developments!

- Ways to engage in development

2 “Seal of Approval”

https://github.com/Rdatatable/data.table/issues/5723

New Developments!

- Ways to engage in development

3 Vote on GitHub Pull Requests

New Developments!

- Ways to engage in development

4 Talk, publish about it

Tyson S. Barrett

Thanks to Matt Dowle and Arun Srinivasan and the data.table team!

 t.barrett88@gmail.com

 tysonbarrett.com

 @healthandstats

 github.com/tysonstanley

 Slides at tysonbarrett.com/teaching